From the proceedings of the *6th Euromicro Workshop on Parallel and Distributed Processing*, Madrid, Spain, Jan. 1998, pp. 3–9. Invited keynote presentation.

High-Performance Mixed-Machine Heterogeneous Computing

Muthucumaru Maheswaran, Tracy D. Braun, and Howard Jay Siegel
Parallel Processing Laboratory, School of Electrical and Computer Engineering
Purdue University, West Lafayette, IN 47907-1285 USA

Abstract

The focus of this invited keynote paper (to be presented by H. J. Siegel) is mixed-machine heterogeneous computing, where a suite of different kinds of high-performance machines are interconnected by high-speed links. Such a system can be orchestrated to perform an application whose subtasks have diverse execution requirements. Subtasks are assigned to and executed on the machines that will result in a minimal overall execution time for the task, considering factors including inter-machine communication overhead. A conceptual model of the automatic decomposition of tasks and assignment of subtasks is presented. Examples of static and dynamic approaches to the matching and scheduling of subtasks are summarized. Some open research problems are discussed.

1 Introduction

Existing high-performance computers typically achieve only a fraction of their peak capabilities on certain portions of some application tasks [8]. This is because different subtasks of an application can have very different computational requirements that result in the need for different machine capabilities. A single machine architecture cannot satisfy all the computational requirements of certain applications equally well. Thus, the use of a heterogeneous computing environment is more appropriate.

The focus of this invited keynote paper (to be presented by H. J. Siegel) is mixed-machine heterogeneous computing (HC), where a suite of different kinds of high-performance machines are interconnected by high-speed links. Such a system provides a variety of architectural capabilities, orchestrated to perform an application whose subtasks have diverse execution requirements [17]. The task must be decomposed into subtasks, where each subtask is computationally homogeneous, and different subtasks may have different machine architectural requirements. These subtasks may share initial or generated data, creating the potential for inter-machine data transfer overhead. To exploit HC systems in such situations, each subtask must be assigned to and executed on the machines that will result in a minimal overall execution time for the task, considering factors including inter-machine communication overhead. The subtasks can be assigned either prior to execution (statically) or during the execution (dynamically).

Figure 1 shows a hypothetical example of an application program whose various subtasks are best suited for execution on different machine architectures [8]. Executing the whole program on a SIMD machine only gives approximately five times the performance achieved by a baseline serial machine. Only the SIMD portion of the program can be executed significantly faster because of the mismatch between each subtask's unique computational requirement and the SIMD architecture. Alternatively, the use of four different machines, each matched to the computational requirements of the subtask to which it was assigned, can result in an execution 50 times as fast as the baseline serial machine.

A conceptual model for automatic HC is introduced in Section 2. As an example of current research in static matching and scheduling, Section 3 presents a genetic-algorithm-based approach. Section 4 describes a dynamic mapping algorithm for on-line matching and scheduling. Open problems are discussed in Section 5.

2 A Conceptual Model for HC

Typically, users of HC systems must perform task decomposition and subtask matching and scheduling themselves (e.g.,[13, 15, 20]). A conceptual model for automatic task decomposition, matching subtasks to machines, and scheduling subtasks is shown in Figure 2. It builds on the model presented in [18] and is referred to as a "conceptual" model because no complete automatic implementation currently exists.

In stage 1, using information about the expected types of application tasks and about the machines in the HC suite, a set of parameters is generated that

This research is supported by the DARPA/ITO Quorum Program under NPS subcontract number N62271-97-M-0900.

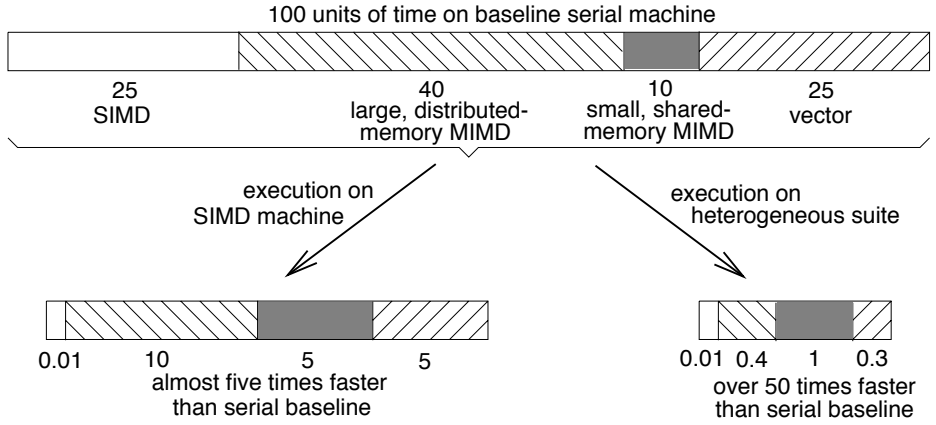

Figure 1: Hypothetical example of the advantage of using a heterogeneous suite of machines, where the heterogeneous suite time includes intermachine communication overhead (based on [8]). Not drawn to scale.

is relevant to both the computational requirements of the applications and the machine capabilities of the HC system. For each parameter, categories for computational characteristics and categories for machine architecture features are derived.

In stage 2, task profiling decomposes the application task into subtasks, each of which is computationally homogeneous. The computational requirements for each subtask are then quantified by profiling the code and data. Analytical benchmarking is used to quantify how effectively each of the available machines in the suite performs on each of the types of computations being considered.

One of the functions of stage 3 is to be able to use the information from stage 2 to derive the estimated execution time of each subtask on each machine in the HC suite and the associated inter-machine communication overhead. Then, these static results, with the machine and inter-machine network initial loading and "status" (i.e., machine/network faults and expected subtask/transfer completion times) are used to generate an assignment of the subtasks to machines and an execution schedule based on certain cost metrics (e.g., minimizing the overall task execution time).

Stage 4 is the execution of the given application. In dynamic matching and scheduling systems, the subtask completion times and loading/status of the machines/network are monitored. This information may be used to reinvoke the matching and scheduling of stage 3 to improve the machine assignment and execution schedule.

Automatic HC is relatively new field. Frameworks for task profiling, analytical benchmarking, and mapping (matching and scheduling) have been proposed, however, further research is needed to make this conceptual model a reality [17, 18].

3 Static Task Mapping

In general, the problem of performing matching and scheduling in an HC environment is NP-complete [5], and therefore some heuristic must be employed. A variety of static approaches have been studied for different HC models (e.g., [4, 11, 16, 19]). As an example of current HC research on mapping statically, a genetic-algorithm approach from [21] is summarized. An application task is decomposed into a set of subtasks S. Let s_i be the i-th subtask. An HC suite consists of a set of machines M. Let m_j be the j-th machine. Each machine can be a different type. The global data items are data items that need to be transferred between subtasks.

The following assumptions about the applications and HC environment are made. Each application task will be represented by a DAG (directed acyclic graph), whose nodes are the subtasks that need to be executed to perform the application and whose arcs are the data dependencies between subtasks. (Note that while the subtasks' dependencies are represented as a DAG, subtasks themselves may contain loops.) Each edge is labeled by the global data item that is transferred over it. The application task has exclusive use of the HC environment, and the genetic-algorithm-based mapper controls the HC machine suite (hardware platform). Subtask execution is non-preemptive. The estimated expected execution time of each subtask on each machine is known. For each pair of machines in the HC suite, an equation for estimating the time to send data between those machines as a function of data set size is known.

Genetic algorithms (GAs) are a heuristic approach

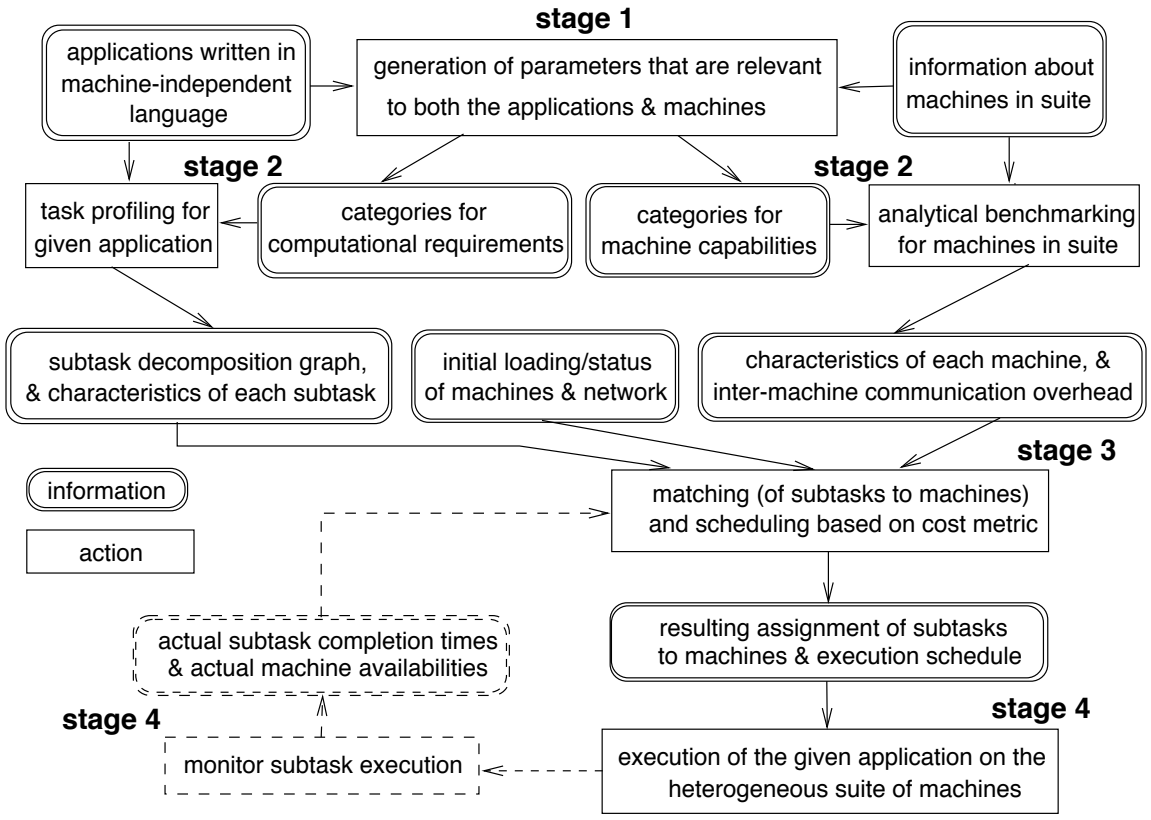

Figure 2: Model for integrating the software support needed for automating the use of heterogeneous computing systems (based on [18]).

to optimization problems that are intractable. The first step is to encode some of the possible solutions as chromosomes, the set of which is referred to as a population. In the [21] approach, each chromosome consists of two parts: the matching string (mat) and the scheduling string (ss). If $\text{mat}(i) = j$, then subtask s_i is assigned to machine m_j. Typically, multiple subtasks will be assigned to the same machine, and then executed in a non-preemptive manner based on an ordering that obeys the precedence constraints (data dependencies) specified in the task DAG. The scheduling string is a topological sort of the DAG representing the task (i.e., a valid total ordering of the partially ordered DAG). If $\text{ss}(k) = i$, then subtask s_i is the k-th subtask in the total ordering. Because it is a topological sort, if subtask $\text{ss}(k)$ needs a global data item created by subtask $\text{ss}(j)$, then $j < k$. The scheduling string gives an ordering of subtasks that is used by the evaluation step.

Each chromosome is associated with a fitness value, which is the completion time of the solution (i.e., mapping) represented by this chromosome (i.e., the expected execution time of the application task if the mapping specified by this chromosome were used). Overlapping among all of the computations and communications performed is limited only by inter-subtask data dependencies and the availability of the machines and the inter-machine network.

In the initial population generation step, a predefined number of distinct chromosomes are randomly created. The solution from a non-evolutionary heuristic is also included in the initial population. After the initial population is determined, the genetic algorithm iterates until a predefined stopping criterion is met. Each iteration consists of the selection, crossover, mutation, and evaluation steps.

In the selection step, some members of the population are removed and others are duplicated. First, all of the chromosomes in the population are ordered (ranked) by their fitness values. Then a rank-based roulette wheel selection scheme is used to implement proportionate selection. The population size is kept constant and a chromosome representing a better solution has a higher probability of having one or more

copies in the next generation population. This GA approach also incorporates elitism, i.e., the best solution found so far is always kept in the population.

The selection step is followed by the crossover step, where some chromosomes are paired and corresponding components of the paired chromosomes are exchanged. The crossover operator for the scheduling strings randomly chooses some pairs of the scheduling strings in the current population. For each pair, it randomly generates a cutoff point, and divides the scheduling strings of the pair into top and bottom parts. Then, the subtasks in each bottom part are reordered using the relative positions of these subtasks in the other original scheduling string, thus guaranteeing that the newly generated scheduling strings are valid schedules. The crossover operator for the matching strings randomly chooses some pairs of the matching strings in the current population. For each pair, it randomly generates a cutoff point, and divides both matching strings of the pair into two parts. Then the machine assignments of the bottom parts are exchanged.

The next step is mutation. The scheduling string mutation operator randomly chooses some scheduling strings in the current population. Then for each chosen scheduling string, it randomly selects a victim subtask. The valid range of the victim subtask is the set of the positions in the scheduling string at which this victim subtask can be placed and still have a valid topological sort of the DAG. The victim subtask is moved randomly to another position in the scheduling string within its valid range. The matching string mutation operator randomly chooses some matching strings in the current population. For each chosen matching string, it randomly selects a subtask entry. Then the machine assignment for the selected entry is changed randomly to another machine.

The last step of an evolution iteration is the evaluation step to determine the fitness value of each chromosome in the current population. A communication subsystem that is modeled after a HiPPI LAN with a central crossbar switch was assumed for the tests that were conducted. As stated earlier, the above steps of selection, crossover, mutation, and evaluation are repeated until one of the stopping criteria are met: (1) the number of iterations reaches some limit (e.g., 1000), (2) the population converged (all the chromosomes had the same fitness value), or (3) the best solution found was not improved after some number of iterations (e.g., 150).

In the tests of this GA approach in [21], simulated program behaviors were used. Small-scale tests were conducted with up to ten subtasks, three machines, and population size 50. For each test, the GA approach found a solution (mapping) that had the same expected completion time for the task as that of the optimal solution found by exhaustive search. Larger tests with up to 100 subtasks, 20 machines, and population size 200 were conducted. This GA approach produced solutions (mappings) that averaged from 150% to 200% better than those produced by the non-evolutionary levelized min-time (LMT) heuristic proposed in [10]. The heuristic in [10] was selected for comparison because it used a similar model of HC. The GA approach took much more time to generate the mappings than did the LMT approach; however, if the mappings are being created off-line, prior to run time, for production tasks that will be executed repeatedly, the generation time is worthwhile.

4 Dynamic Task Mapping

The static mapping algorithms assume that accurate estimates are available for parameters such as subtask completion times. However, in general, such estimates have a degree of uncertainty in them. Therefore, dynamic mapping algorithms that can handle the uncertainty may be needed. Researchers have proposed different dynamic algorithms for varying HC models (e.g., [3, 6, 9, 14]). In dynamic mapping algorithms, machines can come on-line and go off-line at run time.

As an example of current Purdue research on dynamic mapping, an algorithm called the hybrid mapper is discussed in this section. The hybrid mapper uses some results based on an initial static mapping in conjunction with information available only at execution time. It is based on the list scheduling class of algorithms [10]. An initial, statically obtained mapping is provided as input to the hybrid mapper. If the initial mapping is not provided, it should be obtained before running the hybrid mapper.

An HC model similar to the one described in Section 3 is assumed here. Two indices, i and k are associated with each subtask $s_{i,k}$. The index i denotes that $s_{i,k}$ is the i-th subtask in the set S and k is the block number assigned to it by the partitioning algorithm described below. The estimated expected computation time of subtask $s_{i,k}$ on machine m_j is given by $e_{i,k,j}$. The earliest time at which machine m_j is available is given by $A[j]$.

The hybrid mapper executes in two phases. This first phase uses the initial static mapping, expected subtask computation times, and expected data transfer times to preprocess the DAG that represents the application off line. Initially, the DAG is partitioned

into B blocks, numbered consecutively from 0 to $B-1$. The partitioning is done such that the subtasks within a block are independent, i.e., there are no data dependencies among the subtasks in a block. Furthermore, for each block k subtask, $s_{j,k}$, there exists at least one incident edge (data dependency) such that the source subtask is in block $k-1$, i.e., an incident edge from some $s_{i,k-1}$. The $(B-1)$-th block includes the subtasks without any successors and the 0-th block includes only those subtasks without any predecessors.

Once the subtasks in the DAG are partitioned, each subtask is assigned a level by examining the subtasks from block $B-1$ to block 0. The level of each subtask in the $(B-1)$-th block is set to its expected computation time on the machine to which it was assigned by the initial matching. Now consider the k-th block, $0 \leq k < B-1$. Recall $e_{i,k,x}$ is the expected computation time of the subtask $s_{i,k}$ on machine m_x. Let $c_{i,j}$ be the data transfer time for a descendent $s_{j,q}$ of $s_{i,k}$ to get all the relevant data items from $s_{i,k}$ (where $q \geq k+1$). The value of $c_{i,j}$ will be dependent on the machines assigned to subtasks $s_{i,k}$ and $s_{j,q}$ by the initial mapping. Let $\text{level}(s_{i,k})$ be the level of the subtask $s_{i,k}$. Let $\text{iss}(s_{i,k})$ be the immediate successor set of subtask $s_{i,k}$ such that there is an arc from $s_{i,k}$ to each member of $\text{iss}(s_{i,k})$ in the DAG. With these definitions, the level of a subtask $s_{i,k}$ that is initially assigned to m_x is given by:

$$\text{level}(s_{i,k}) = e_{i,k,x} + \max_{s_{j,q} \in \text{iss}(s_{i,k})} (c_{i,j} + \text{level}(s_{j,q})).$$

The level of a subtask can be interpreted as the length of the critical path from the point the given subtask is located on the DAG to a subtask with no successors. The hybrid mapper is based on the heuristic idea that by executing the subtasks with higher levels as quickly as possible, the overall expected completion time for the application can be minimized.

The second phase of the hybrid mapper involves the actual execution of the subtasks. The execution of the subtasks proceeds from block 0 to block $B-1$. A block k is considered to be executing if at least one subtask from block k is executing. The execution of several blocks can overlap with each other in time, i.e., subtasks from different blocks could be executing at the same time.

The hybrid mapper starts remapping the block k subtasks when the first block $(k-1)$ subtask begins its execution. When block k is being scheduled, it is highly likely that actual execution time information can be used for many subtasks from blocks 0 to $k-2$. There may be some subtasks from blocks 0 to $k-2$ that could still be executing or awaiting execution when subtasks from block k are being considered for remapping. For such subtasks, expected execution times are used.

In a list-scheduling type of algorithm, the subtasks are first ordered based on some priority. Then, each subtask is mapped by examining the list of subtasks from the highest priority subtask to the lowest priority subtask. The machine to which each subtask is assigned depends on the matching criterion used by the particular algorithm. In the hybrid mapper, the priority of a subtask is equal to the level of that subtask that was computed statically in the first phase. The matching criterion used for subtask $s_{i,k}$ is the minimization of the partial completion time, defined below.

Let m_x be the machine on which $s_{i,k}$ is being considered for execution. Then let $\text{pct}(s_{i,j}, x)$ denote the partial completion time of the subtask $s_{i,k}$ on machine m_x, $\text{dr}(s_{i,k})$ be the time at which the last data item required by $s_{i,k}$ to begin its execution arrives at m_x, and $\text{ips}(s_{i,k})$ be the immediate predecessor set for subtask $s_{i,k}$ such that there is an arc to $s_{i,k}$ from each member of $\text{ips}(s_{i,k})$ in the DAG. For any subtask $s_{i,k}$, where $s_{j,q} \in \text{ips}(s_{i,k})$, and $s_{j,q}$ is currently mapped onto machine m_y,

$$\begin{aligned} \text{dr}(s_{i,k}) &= \max_{s_{j,q} \in \text{ips}(s_{i,k})} (c_{j,i} + \text{pct}(s_{j,q}, y)), \\ \text{pct}(s_{i,k}, x) &= e_{i,k,x} + \max(A[x], \text{dr}(s_{i,k})). \end{aligned}$$

The subtask $s_{i,k}$ is remapped onto the machine m_x that gives the minimum $\text{pct}(s_{i,k}, x)$, and $A[x]$ is updated using $\text{pct}(s_{i,k})$. Then the next subtask from the list is considered for mapping.

The simulation results indicate that the performance of a statically obtained initial mapping can be improved by the hybrid mapper. Initial mappings were generated using the baseline algorithm [21]. The baseline algorithm partitions the subtasks into blocks using an algorithm similar to the one described here. Once the subtasks are partitioned into blocks, they are ordered such that a subtask in block k comes before a subtask in block l, where $k < l$. The subtasks within a block are arranged such that the subtasks with more descendents appear before the subtasks with less descendents (ties are broken arbitrarily). The subtasks are considered for assignment by traversing the list, beginning with block 0 subtasks. A subtask is mapped to the machine that gives the shortest partial completion time for the subtasks that have been mapped (including this subtask).

From the simulation results obtained, performance

improvement from using the hybrid mapper can be as much as 15% for some cases. The timings also indicate that the remapping time needed per block of subtasks is in the order hundreds of milliseconds for up to 50 machines and 500 subtasks. In the worst case situation, to obtain complete overlap between the execution of the subtasks and the operation of the hybrid mapper, the computation time for the shortest running subtask must be greater than the per block remapping time. Ongoing research will examine ways to increase the performance gain obtained from the use of the hybrid remapper.

5 Open Research Problems

There are a great many open problems that need to be solved before HC can be made available to application programmers in a transparent way (summarized here from [12, 17, 18]). Implementation of the automatic HC programming environment envisioned in the conceptual model in Section 2 will require a great deal of research for devising practical and theoretically sound methodologies for each component of every stage. A general question that is particularly applicable to stages 1 and 2 of the conceptual model is: "What information should (must) the user provide and what information should (can) be determined automatically?"

To program an HC system, it would be best to have one or more machine-independent programming languages [22] that allow the user to augment the code with compiler directives. The language and directives should be designed to facilitate (a) the compilation of the program into efficient code for the machines in the suite, (b) the task decomposition, (c) the determination of computational requirements of each subtask, and (d) the use of machine-dependent subroutine libraries.

There is a need for debugging and performance tuning tools that can be used across an HC suite of machines. This involves research in the areas of distributed programming environments and visualization techniques.

Ideally, information about the current loading and status of the machines in the HC suite and the network should be incorporated into the mapping decisions. Methods must be developed for measuring the current loading, determining the status (e.g., faulty or not), and estimating the subtask completion times. Also, the uncertainty present in the estimated parameter values such as subtask completion times should be taken into consideration in determining the machine assignment and execution schedule.

There is much ongoing research in the area of inter-machine data transport. This research includes the hardware support required, the software protocols required, designing the network topology, computing the minimum-time path between two machines, and devising rerouting schemes in case of faults or heavy loads. Related to this is the data formatting problem, involving issues such as data type storage formats and sizes, byte ordering within data types, and machines' network-interface buffer sizes.

Another area of research is dynamic task migration between different parallel machines at execution time. Current research in this area involves determining how to move an executing task between different machines [1, 2] and how to use dynamic task migration for load rebalancing or fault tolerance.

Some of the future research outlined here may be pursued as part of a DARPA/ITO Quorum Program project called MSHN (Management System for Heterogeneous Networks). MSHN is a collaborative research effort among NPS (Naval Postgraduate School), NRaD (a Naval Laboratory), Purdue, and USC (University of Southern California). It builds on SmartNet, an operational scheduling framework and system for managing resources in a heterogeneous environment developed at NRaD [7]. The technical objective of the MSHN project is to design, prototype, and refine a distributed resource management system that leverages the heterogeneity of resources and tasks to deliver the requested qualities of service.

In summary, while the use of existing HC systems demonstrates their significant benefits, the amount of effort currently required to implement an application on an HC system can be substantial. Future research on the above open problems will improve this situation and allow HC to realize its inherent potential.

Acknowledgments - A version of some of this material also appeared in an invited keynote paper (by H. J. Siegel and M. Maheswaran) at the IX Brazilian Symposium on Computer Architectures – High Performance Computing.

References

[1] J. B. Armstrong, H. J. Siegel, W. Cohen, M. Tan, H. G. Dietz, and J. A. B. Fortes, "Dynamic task migration from SPMD to SIMD virtual machines," *1994 Int'l Conf. Parallel Processing (ICPP '94)*, Vol. II, Aug. 1994, pp. 160-169.

[2] J. B. Armstrong and H. J. Siegel, "Dynamic task migration from SIMD to SPMD virtual machines," *1st IEEE Int'l Conf. Engineering of Complex Computer Systems*, Nov. 1995, pp. 326-333.

[3] J. R. Budenske, R. S. Ramanujan, and H. J. Siegel, "A method for the on-line use of off-line derived remappings of iterative automatic target recognition tasks onto a particular class of heterogeneous parallel platforms," *Parallel Computing*, to appear in 1998 (preliminary version in *6th Heterogeneous Computing Workshop (HCW '97)*, Apr. 1997, pp. 96-110).

[4] M. M. Eshaghian, ed., *Heterogeneous Computing*, Artech House, Norwood, MA, 1996.

[5] D. Fernandez-Baca, "Allocating modules to processors in a distributed system," *IEEE Trans. Software Engineering*, Vol. SE-15, No. 11, Nov. 1989, pp. 1427-1436.

[6] R. F. Freund, B. R. Carter, D. Watson, E. Keith, and F. Mirabile, "Generational scheduling for heterogeneous computing systems," *Int'l Conf. Parallel and Distributed Processing Techniques and Applications (PDPTA '96)*, Aug. 1996, pp. 769-778.

[7] R. F. Freund, T. Kidd, D. Hensgen, and L. Moore, "SmartNet: A scheduling framework for meta-computing," *2nd Int'l Symp. Parallel Architectures, Algorithms, and Networks (ISPAN '96)*, June 1996, pp. 514-521.

[8] R. F. Freund and H. J. Siegel, "Heterogeneous processing," *IEEE Computer*, Vol. 26, No. 6, June 1993, pp. 13-17.

[9] B. Hamidzadeh, D. J. Lilja, and Y. Atif, "Dynamic scheduling techniques for heterogeneous computing systems," *Concurrency: Practice and Experience*, Vol. 7, No. 7, Oct. 1995, pp. 633-652.

[10] M. A. Iverson, F. Ozguner, and G. J. Follen, "Parallelizing existing applications in a distributed heterogeneous environment," *4th Heterogeneous Computing Workshop (HCW '95)*, Apr. 1995, pp. 93-100.

[11] M. Kafil and I. Ahmad, "Optimal task assignment in heterogeneous computing systems," *6th Heterogeneous Computing Workshop (HCW '97)*, Apr. 1997, pp. 135-146.

[12] A. Khokhar, V. K. Prasanna, M. Shaaban, and C. L. Wang, "Heterogeneous computing: Challenges and opportunities," *IEEE Computer*, Vol. 26, No. 6, June 1993, pp. 18-27.

[13] A. E. Klietz, A. V. Malevsky, and K. Chin-Purcell, "A case study in metacomputing: Distributed simulations of mixing in turbulent convection," *2nd Workshop on Heterogeneous Processing (WHP '93)*, Apr. 1993, pp. 101-106.

[14] C. Leangsuksun, J. Potter, and S. Scott, "Dynamic task mapping algorithms for a distributed heterogeneous computing environment," *4th Heterogeneous Computing Workshop (HCW '95)*, Apr. 1995, pp. 30-34.

[15] J. Rosenman and T. Cullip, "High-performance computing in radiation cancer treatment," *CRC Critical Reviews in Biomedical Engineering*, Vol. 20, 1992, pp. 391-402.

[16] P. Shroff, D. W. Watson, N. S. Flann, and R. F. Freund, "Genetic simulated annealing for scheduling data-dependent tasks in heterogeneous environments," *5th Heterogeneous Computing Workshop (HCW '96)*, Apr. 1996, pp. 98-117.

[17] H. J. Siegel, J. K. Antonio, R. C. Metzger, M. Tan, and Y. A. Li, "Heterogeneous computing," in *Parallel and Distributed Computing Handbook*, A. Y. Zomaya, ed., McGraw-Hill, New York, NY, 1996, pp. 725-761.

[18] H. J. Siegel, H. G. Dietz, and J. K. Antonio, "Software support for heterogeneous computing," in *The Computer Science and Engineering Handbook*, A. B. Tucker, Jr., ed., CRC Press, Boca Raton, FL, 1997, pp. 1886-1909.

[19] H. Singh and A. Youssef, "Mapping and scheduling heterogeneous task graphs using genetic algorithms," *5th Heterogeneous Computing Workshop (HCW '96)*, Apr. 1996, pp. 86-97.

[20] "Special report: Gigabit network testbeds," *IEEE Computer*, Vol. 23, No. 9, Sep. 1990, pp. 77-80.

[21] L. Wang, H. J. Siegel, V. P. Roychowdhury, and A. A. Maciejewski. "Task matching and scheduling in heterogeneous computing environments using a genetic-algorithm-based approach," *J. Parallel and Distributed Computing*, to appear in 1998 (preliminary version in *5th Heterogeneous Computing Workshop (HCW '96)*, Apr. 1996, pp. 72-85).

[22] C. C. Weems, G. E. Weaver, and S. G. Dropsho, "Linguistic support for heterogeneous parallel processing: a survey and an approach," *3rd Heterogeneous Computing Workshop (HCW '94)*, Apr. 1994, pp. 81-88.